Summary of the Catalog Manager

C Summary

Constants and Data Types

```
enum {
   kThisRecordOwnerBit           =  0,
   kFriendsBit                   =  1,
   kAuthenticatedInDNodeBit      =  2,
   kAuthenticatedInDirectoryBit  =  3,
   kGuestBit                     =  4,
   kMeBit                        =  5
};

enum {         /* Values of CategoryMask */
   kThisRecordOwnerMask          = (1L << kThisRecordOwnerBit),
   kFriendsMask                  = (1L << kFriendsBit),
   kAuthenticatedInDNodeMask     = (1L << kAuthenticatedInDNodeBit),
   kAuthenticatedInDirectoryMask = (1L << kAuthenticatedInDirectoryBit),
   kGuestMask                    = (1L << kGuestBit),
   kMeMask                       = (1L << kMeBit)

typedef unsigned long CategoryMask;

};enum {
      kEnumDistinguishedNameBit,
      kEnumAliasBit,
      kEnumPseudonymBit,
      kEnumDNodeBit,
      kEnumInvisibleBit
};

enum {
      /* values of DirEnumChoices */
      kEnumDistinguishedNameMask = 1L<<kEnumDistinguishedNameBit,
      kEnumAliasMask             = 1L<<kEnumAliasBit,
      kEnumPseudonymMask         = 1L<<kEnumPseudonymBit,
```

```
      kEnumDNodeMask             = 1L<<kEnumDNodeBit,
      kEnumInvisibleMask         = 1L<<kEnumInvisibleBit
};

#define kEnumAllMask (kEnumDistinguishedNameMask | kEnumAliasMask |
                      kEnumPseudonymMask | kEnumDNodeMask |
                      EnumInvisibleMask)

typedef unsigned long DirEnumChoices;

/* values of DirSortOption */
enum {
      kSortByName= 0,
      kSortByType= 1
};

typedef unsigned short DirSortOption;

/* values of DirSortDirection */

enum {
      kSortForwards= 0,
      kSortBackwards= 1
};

typedef unsigned short DirSortDirection;

/* values of DirMatchWith */
enum {
      kMatchAll,
      kExactMatch,
      kBeginsWith,
      kEndingWith,
      kContaining
};

typedef unsigned char DirMatchWith;

#define kCurrentOCESortVersion11

enum {
      kSupportsDNodeNumberBit
      kSupportsRecordCreationIDBit
      kSupportsAttributeCreationIDBit
      kSupportsMatchAllBit
      kSupportsBeginsWithBit
```

```
        kSupportsExactMatchBit
        kSupportsEndsWithBit
        kSupportsContainsBit
        kSupportsOrderedEnumerationBit
        kCanSupportNameOrderBit
        kCanSupportTypeOrderBit
        kSupportsSortBackwardsBit
        kSupportIndexRatioBit
        kSupportsEnumerationContinueBit
        kSupportsLookupContinueBit
        kSupportsEnumerateAttributeTypeContinueBit
        kSupportsEnumeratePseudonymContinueBit
        kSupportsAliasesBit
        kSupportsPseudonymsBit
        kSupportsPartialPathnamesBit
        kSupportsAuthenticationBit
        kSupportsProxiesBit
        kSupportsFindRecordBit
};

/* values of DirGestalt` */
enum {
        kSupportsDNodeNumberMask         = 1L<<kSupportsDNodeNumberBit,
        kSupportsRecordCreationIDMask    = 1L<<kSupportsRecordCreationIDBit,
        kSupportsAttributeCreationIDMask = 1L<<kSupportsAttributeCreationIDBit,
        kSupportsMatchAllMask            = 1L<<kSupportsMatchAllBit,
        kSupportsBeginsWithMask          = 1L<<kSupportsBeginsWithBit,
        kSupportsExactMatchMask          = 1L<<kSupportsExactMatchBit,
        kSupportsEndsWithMask            = 1L<<kSupportsEndsWithBit,
        kSupportsContainsMask            = 1L<<kSupportsContainsBit,
        kSupportsOrderedEnumerationMask  = 1L<<kSupportsOrderedEnumerationBit,
        kCanSupportNameOrderMask         = 1L<<kCanSupportNameOrderBit,
        kCanSupportTypeOrderMask         = 1L<<kCanSupportTypeOrderBit,
        kSupportSortBackwardsMask        = 1L<<kSupportSortBackwardsBit,
        kSupportIndexRatioMask           = 1L<<kSupportIndexRatioBit,
        kSupportsEnumerationContinueMask = 1L<<kSupportsEnumerationContinueBit,
        kSupportsLookupContinueMask      = 1L<<kSupportsLookupContinueBit,
        kSupportsEnumerateAttributeTypeContinueMask =
                                1L<<kSupportsEnumerateAttributeTypeContinueBit,
        kSupportsEnumeratePseudonymContinueMask =
                                1L<<kSupportsEnumeratePseudonymContinueBit,
        kSupportsAliasesMask             = 1L<<kSupportsAliasesBit,
        kSupportsPseudonymsMask          = 1L<<kSupportsPseudonymsBit,
        kSupportsPartialPathNamesMask    = 1L<<kSupportsPartialPathNamesBit,
```

```
      kSupportsAuthenticationMask      = 1L<<kSupportsAuthenticationBit,
      kSupportsProxiesMask             = 1L<<kSupportsProxiesBit,
      kSupportsFindRecordMask          = 1L<<kSupportsFindRecordBit
};

typedef unsigned long DirGestalt;

struct DNodeID {
   DNodeNum    dNodeNumber;   /* dNode number  */
   long        reserved1;     /* reserved */
   RStringPtr  name;          /* name of the dNode */
   long        reserved2;     /* reserved */
};

typedef struct DNodeID DNodeID;

struct DirEnumSpec {
   DirEnumChoices    enumFlag;
   unsigned short    indexRatio; /* if supported, record position between 1
                                    and 100. 0 if not supported */
   union {
      LocalRecordID  recordIdentifier;
      DNodeID        dNodeIdentifier;
   }u;
};

typedef struct DirEnumSpec DirEnumSpec;

struct DirMetaInfo {
   unsigned longinfo[4];
};

typedef struct DirMetaInfo DirMetaInfo;

struct SLRV {
   ScriptCode  script;     /* script code in which entries are sorted */
   short       language;   /* language code in which entries are sorted */
   short       regionCode; /* region code in which entries are sorted */
   short       version;    /* version of AOCE sorting software */
};

typedef struct SLRV SLRV;

typedef unsigned long AuthIdentity;
```

```
typedef pascal Boolean (*ForEachRecordID) (long clientData,
                                           const RecordID* recordID);

typedef pascal Boolean (*ForEachAttrType) (long clientData,
                                           const AttributeType *attrType);

{ FUNCTION ForEachLookupRecordID(clientData: long; recordID: RecordID): BOOL-
EAN;}

{ FUNCTION ForEachAttrTypeLookup(clientData: long; attrType:
AttributeTypePtr; myAttrAccMask: AccessMask): BOOLEAN;}

{ FUNCTION ForEachAttrValue(clientData: long; attribute: Attribute):
BOOLEAN;}

typedef pascal Boolean (*ForEachDNodeAccessControl) (long clientData,
                        const DSSpec *dsObj, AccessMask activeDnodeAccMask,
                        AccessMask defaultRecordAccMask,
                        AccessMask defaultAttributeAccMask);

#define  AuthDirParamHeader
   Ptr            qLink;         /* reserved */\
   long           reserved_H1;   /* reserved */\
   long           reserved_H2;   /* reserved */\
   ProcPtr        ioCompletion;  /* your completion routine */\
   OSErr          ioResult;      /* result code */\
   unsigned long  saveA5;        /* reserved */\
   short          reqCode;       /* CSAM request code*/\
   long           reserved[2];   /* reserved */\
   AddrBlock      serverHint;    /* PowerShare server's AppleTalk address */
   short          dsRefNum;      /* personal catalog reference number */\
   unsigned long  callID;        /* reserved */\
   AuthIdentity   identity;      /* requester's authentication identity */
   long           gReserved1;    /* reserved */\
   long           gReserved2;    /* reserved */\
   long           gReserved3;    /* reserved */\
   long           clientData;    /* you define this field */

struct DirEnumerateDirectoriesGetPB {
   AuthDirParamHeader
   OCEDirectoryKind  directoryKind;            /* enumerate catalogs
                                                  bearing this signature */
   DirectoryNamePtr  startingDirectoryName;    /* starting catalog */
   DirDiscriminator  startingDirDiscriminator; /* starting catalog
                                                  discriminator */
```

```
   long              eReserved;
   long              fReserved;
   long              gReserved;
   long              hReserved;
   Boolean           includeStartingPoint;      /* if true, return the
                                                   catalog specified by
                                                   starting point */
   Byte              padByte;
   short             i1Reserved;
   Ptr               getBuffer;
   unsigned long     getBufferSize;
};

typedef struct DirEnumerateDirectoriesGetPB DirEnumerateDirectoriesGetPB;

struct DirEnumerateDirectoriesParsePB {
   AuthDirParamHeader
   long              aReserved;
   long              bReserved;
   long              cReserved;
   long              dReserved;
   ForEachDirectory  eachDirectory;
   long              fReserved;
   long              gReserved;
   long              hReserved;
   long              iReserved;
   Ptr               getBuffer;
   unsigned long     getBufferSize;
};

typedef struct DirEnumerateDirectoriesParsePB DirEnumerateDirectoriesParsePB;

struct DirFindRecordGetPB {
   AuthDirParamHeader
   RecordIDPtr       startingPoint;
   long              reservedA[2];
   RStringPtr        nameMatchString;
   RStringPtr*       typesList;
   unsigned long     typeCount;
   long              reservedB;
   short             reservedC;
   DirMatchWith      matchNameHow;
   DirMatchWith      matchTypeHow;
   Ptr               getBuffer;
```

```
   unsigned long     getBufferSize;
   DirectoryNamePtr  directoryName;
   DirDiscriminator  discriminator;
};

typedef struct DirFindRecordGetPB DirFindRecordGetPB;

struct DirFindRecordParsePB {
   AuthDirParamHeader
   RecordIDPtr       startingPoint;
   long              reservedA[2];
   RStringPtr        nameMatchString;
   RStringPtr*       typesList;
   unsigned long     typeCount;
   long              reservedB;
   short             reservedC;
   DirMatchWith      matchNameHow;
   DirMatchWith      matchTypeHow;
   Ptr               getBuffer;
   unsigned long     getBufferSize;
   DirectoryNamePtr  directoryName;
   DirDiscriminator  discriminator;
   ForEachRecord     forEachRecordFunc;
};

typedef struct DirFindRecordParsePB DirFindRecordParsePB;

struct DirGetDirectoryInfoPB {
   AuthDirParamHeader
   DirectoryNamePtr  directoryName;    /* catalog name */

   DirDiscriminator  discriminator; /* descriminate between duplicate
                                       catalog names */
   DirGestalt        features;      /* capability bit flags */
};

typedef struct DirGetDirectoryInfoPB DirGetDirectoryInfoPB;

struct DirGetLocalNetworkSpecPB {
   AuthDirParamHeader
   DirectoryNamePtr  directoryName;    /* catalog name */
   DirDiscriminator  discriminator;    /* discriminator */
   NetworkSpecPtr    networkSpec;      /* NetworkSpec */
};
```

```
typedef struct DirGetLocalNetworkSpecPB DirGetLocalNetworkSpecPB;

struct DirGetDirectoryIconPB {
   AuthDirParamHeader
   PackedRLIPtr      pRLI;            /* packed RLI for the catalog */
   OSType            iconType;        /* type of icon requested */
   Ptr               iconBuffer;      /* buffer to hold icon data */
   unsigned long     bufferSize;      /* size of buffer to hold icon data */
};

typedef struct DirGetDirectoryIconPB DirGetDirectoryIconPB;

struct DirGetExtendedDirectoriesInfoPB {
   AuthDirParamHeader
   Ptr               buffer;          /* Pointer to a buffer where data
                                         will be returned */
   unsigned long     bufferSize;         length of actual data will be
                                         returned here */
   unsigned long     totalEntries;    /* total number of catalogs found */
   unsigned long     actualEntries;   /* total number of catalog entries */
                                         returned */
};

typedef struct DirGetExtendedDirectoriesInfoPB
                                         DirGetExtendedDirectoriesInfoPB;

typedef pascal Boolean (*ForEachDirectory) (
   long clientData, const DirectoryName *dirName,
   const DirDiscriminator *discriminator,DirGestalt features);

struct DirEnumerateGetPB {
   AuthDirParamHeader
   PackedRLIPtr      aRLI;            /* an RLI specifying the cluster
                                         to be enumerated */
   DirEnumSpec       *startingPoint;
   DirSortOption     sortBy;
   DirSortDirection  sortDirection;
   long              dReserved;
   RStringPtr        nameMatchString; /* name from which enumeration should
                                         start */
   RStringPtr        *typesList;      /* list of entity types to be
                                         enumerated */
   unsigned long     typeCount;       /*  number of types in the list */
   DirEnumChoices    enumFlags;       /*  indicates what to enumerate */
   Boolean           includeStartingPoint;
```

```
                                        /* if true, return the record
                                           specified in starting point */
   Byte              padByte;
   DirMatchWith      matchNameHow;      /* matching criteria for
                                           nameMatchString */
   DirMatchWith      matchTypeHow;      /* matching riteria for typeList */
   Ptr               getBuffer;
   unsigned long     getBufferSize;
   SLRV              responseSLRV;      /* response SLRV */
};c

typedef struct DirEnumerateGetPB DirEnumerateGetPB;

struct DirEnumerateParsePB {
   AuthDirParamHeader
   PackedRLIPtr         aRLI;          /* an RLI specifying the cluster to
                                           be enumerated */
   long                 bReserved;
   long                 cReserved;
   ForEachDirEnumSpec   eachEnumSpec;
   long                 eReserved;
   long                 fReserved;
   long                 gReserved;
   long                 hReserved;
   long                 iReserved;
   Ptr                  getBuffer;
   unsigned long        getBufferSize;
   short                l1Reserved;
   short                l2Reserved;
   short                l3Reserved;
   short                l4Reserved;

};

typedef struct DirEnumerateParsePB DirEnumerateParsePB;

struct DirGetDNodeMetaInfoPB {
   AuthDirParamHeader
   PackedRLIPtr   pRLI;
   DirMetaInfo    metaInfo;
};

typedef struct DirGetDNodeMetaInfoPB DirGetDNodeMetaInfoPB;
```

```
struct DirMapDNodeNumberToPathNamePB {
   AuthDirParamHeader
   DirectoryNamePtr  directoryName;       /* catalog name */
   DirDiscriminator  discriminator;       /* discriminator */
   DNodeNum          dNodeNumber;         /* dNode number to be mapped */
   PackedPathNamePtr path;                /* packed pathname returned */
   unsigned short    lengthOfPathName;
                                          /* length of packed pathname
                                             structure*/
};

typedef struct DirMapDNodeNumberToPathNamePB DirMapDNodeNumberToPathNamePB;

struct DirMapPathNameToDNodeNumberPB {
   AuthDirParamHeader
   DirectoryNamePtr     directoryName;    /* catalog name */
   DirDiscriminator     discriminator;    /* discriminator */
   DNodeNum             dNodeNumber;      /* dNode number to the path */
   PackedPathNamePtr    path;             /* pathname to be mapped */
};

typedef struct DirMapPathNameToDNodeNumberPB DirMapPathNameToDNodeNumberPB;

struct DirGetDNodeInfoPB {
   AuthDirParamHeader
   PackedRLIPtr   pRLI;             /* packed RLI whose info is requested */
   DirNodeKind    descriptor;       /* dNode descriptor */
   NetworkSpecPtr networkSpec;      /* cluster's networkSpec if kIsCluster */
};

typedef struct DirGetDNodeInfoPB DirGetDNodeInfoPB;

struct DirAddADAPDirectoryPB {
   AuthDirParamHeader
   DirectoryNamePtr  directoryName;    /* catalog name */
   DirDiscriminator  discriminator;    /* discriminate between duplicate
                                          catalog names */
   Boolean           addToOCESetup;    /* add this catalog to PowerTalk
                                          Setup */
   Byte              padByte;
   CreationID        directoryRecordCID;
                                       /* creation ID for the catalog
                                          record */
};
```

```
typedef struct DirAddADAPDirectoryPB DirAddADAPDirectoryPB;

struct DirFindADAPDirectoryByNetSearchPB {
   AuthDirParamHeader
   DirectoryNamePtr  directoryName;    /* catalog name */
   DirDiscriminator  discriminator;    /* discriminate between duplicate
                                          catalog names */
   Boolean           addToOCESetup;    /* add this catalog to PowerTalk
                                          setup list */
   Byte              padByte;
   CreationID        directoryRecordCID;
                                       /* creation ID for the catalog
                                          record */
};

typedef struct DirFindADAPDirectoryByNetSearchPB
                                          DirFindADAPDirectoryByNetSearchPB;

struct DirNetSearchADAPDirectoriesGetPB {
   AuthDirParamHeader
   Ptr            getBuffer;
   unsigned long  getBufferSize;
   long           cReserved;
};

typedef struct DirNetSearchADAPDirectoriesGetPB
                                          DirNetSearchADAPDirectoriesGetPB;

struct DirNetSearchADAPDirectoriesParsePB {
   AuthDirParamHeader
   Ptr                     getBuffer;
   unsigned long           getBufferSize;
   ForEachADAPDirectory    eachADAPDirectory;
};

typedef struct DirNetSearchADAPDirectoriesParsePB
                                       DirNetSearchADAPDirectoriesParsePB;

typedef pascal Boolean (*ForEachADAPDirectory) (
   long clientData, const DirectoryName *dirName,
   const DirDiscriminator *discriminator, DirGestalt features,
   AddrBlock serverHint);
```

```
struct DirRemoveDirectoryPB {
   AuthDirParamHeader
   CreationID  directoryRecordCID;  /* creation ID for the catalog record */
};

typedef struct DirRemoveDirectoryPB DirRemoveDirectoryPB;

struct DirGetOCESetupRefNumPB {
   AuthDirParamHeader
   CreationID  oceSetupRecordCID;/* creation ID for the catalog record */
};

typedef struct DirGetOCESetupRefNumPB DirGetOCESetupRefNumPB;

struct DirCreatePersonalDirectoryPB {
   AuthDirParamHeader
   FSSpecPtr   fsSpec;     /* FSSpec for the personal catalog */
   OSType      fdType;     /* file type for the personal catalog */
   OSType      fdCreator;  /* file creator for the personal catalog */
};

typedef struct DirCreatePersonalDirectoryPB DirCreatePersonalDirectoryPB;

struct DirOpenPersonalDirectoryPB {
   AuthDirParamHeader
   FSSpecPtr   fsSpec;           /* open an existing personal catalog */
   char        accessRequested;  /* open: permissions requested(byte) */
   char        accessGranted;    /* open: permissions (byte) (granted) */
   DirGestalt  features;         /* features for personal catalog */
};

typedef struct DirOpenPersonalDirectoryPB DirOpenPersonalDirectoryPB;

struct DirClosePersonalDirectoryPB {
   AuthDirParamHeader
};

typedef struct DirClosePersonalDirectoryPB DirClosePersonalDirectoryPB;

struct DirMakePersonalDirectoryRLIPB {
   AuthDirParamHeader
   FSSpecPtr      fromFSSpec;       /* FSSpec for creating relative alias */
   unsigned short pRLIBufferSize;   /* length of 'pRLI' buffer */
   unsigned short pRLISize;         /* length of actual 'pRLI' */
   PackedRLIPtr   pRLI;             /* pRLI for the specified address book */
};
```

```
typedef struct DirMakePersonalDirectoryRLIPB DirMakePersonalDirectoryRLIPB;

struct DirAddRecordPB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;          /* Creation ID returned here */
   Boolean        allowDuplicate;
};

typedef struct DirAddRecordPB DirAddRecordPB;

struct DirDeleteRecordPB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
};

typedef struct DirDeleteRecordPB DirDeleteRecordPB;

struct DirGetRecordMetaInfoPB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
   DirMetaInfo    metaInfo;
};

typedef struct DirGetRecordMetaInfoPB DirGetRecordMetaInfoPB;

struct DirGetNameAndTypePB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
};

typedef struct DirGetNameAndTypePB DirGetNameAndTypePB;

struct DirSetNameAndTypePB {
   AuthDirParamHeader
   RecordIDPtr aRecord;
   Boolean     allowDuplicate;
   Byte        padByte;
   RStringPtr  newName;          /* new name for the record */
   RStringPtr  newType;          /* new type for the record */
};

typedef struct DirSetNameAndTypePB DirSetNameAndTypePB;

struct DirAddPseudonymPB {
   AuthDirParamHeader
   RecordIDPtr aRecord;          /* Record ID to be added to pseudonym */
```

```
   RStringPtr  pseudonymName;    /* new name to be added as pseudonym */
   RStringPtr  pseudonymType;    /* new name to be added as pseudonym */
   Boolean     allowDuplicate;
};

typedef struct DirAddPseudonymPB DirAddPseudonymPB;

struct DirDeletePseudonymPB {
   AuthDirParamHeader
   RecordIDPtr aRecord;       /* Record ID to which pseudonym is
                                 to be added */
   RStringPtr  pseudonymName; /* pseudonym name to be deleted */
   RStringPtr  pseudonymType; /* pseudonym type to be deleted */
};

typedef struct DirDeletePseudonymPB DirDeletePseudonymPB;

struct DirEnumeratePseudonymGetPB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
   RStringPtr     startingName;
   RStringPtr     startingType;
   long           dReserved;
   long           eReserved;
   long           fReserved;
   long           gReserved;
   long           hReserved;
   Boolean        includeStartingPoint;   /* if true, the pseudonym
                                             specified by starting point will
                                             be included */
   Byte           padByte;
   short          i1Reserved;
   Ptr            getBuffer;
   unsigned long  getBufferSize;
};

typedef struct DirEnumeratePseudonymGetPB DirEnumeratePseudonymGetPB;

struct DirEnumeratePseudonymParsePB {
   AuthDirParamHeader
   RecordIDPtr       aRecord;       /* same as DirEnumerateAliasesGetPB */
   long              bReserved;
   long              cReserved;
   ForEachRecordID   eachRecordID;
   long              eReserved;
```

```
   long              fReserved;
   long              gReserved;
   long              hReserved;
   long              iReserved;
   Ptr               getBuffer;
   unsigned long     getBufferSize;
};

typedef struct DirEnumeratePseudonymParsePB DirEnumeratePseudonymParsePB;

struct DirAddAliasPB {
   AuthDirParamHeader
   RecordIDPtr aRecord;
   Boolean     allowDuplicate;
};

typedef struct DirAddAliasPB DirAddAliasPB;

struct DirAddAttributeValuePB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
   AttributePtr   attr;    /* AttributeCreationID returned here */
};

typedef struct DirAddAttributeValuePB DirAddAttributeValuePB;

struct DirDeleteAttributeValuePB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
   AttributePtr   attr;
};

typedef struct DirDeleteAttributeValuePB DirDeleteAttributeValuePB;

struct DirChangeAttributeValuePB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
   AttributePtr   currentAttr;
   AttributePtr   newAttr;
};
#ifndef __cplusplus
typedef struct DirChangeAttributeValuePB DirChangeAttributeValuePB;
#endif
```

```
struct DirVerifyAttributeValuePB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
   AttributePtr   attr;
};

typedef struct DirVerifyAttributeValuePB DirVerifyAttributeValuePB;

struct DirFindValuePB {
   AuthDirParamHeader
   PackedRLIPtr      aRLI;               /* an RLI specifying the cluster to
                                            be enumerated */
   LocalRecordIDPtr  aRecord;            /* if not nil, look only in this
                                            record */
   AttributeTypePtr  attrType;           /* if not nil, look only in this
                                            attribute type */
   LocalRecordIDPtr  startingRecord;     /* record in which to start
                                            searching */
   AttributePtr      startingAttribute;  /* attribute in which to start
                                            searching */
   LocalRecordIDPtr  recordFound;        /* record in which data was
                                            found */
   Attribute         attributeFound;     /* attribute in which data was
                                            found */
   unsigned long     matchSize;          /* length of matching bytes */
   Ptr               matchingData;       /* data bytes to be matched in */
                                            search */
   DirSortDirection  sortDirection;      /* sort direction (forward or */
                                            backward) */
};

typedef struct DirFindValuePB DirFindValuePB;

struct DirLookupGetPB {
   AuthDirParamHeader
   RecordIDPtr       *aRecordList;  /* an array of record ID pointers */
   AttributeTypePtr  *attrTypeList; /* an array of attribute types */
   long              cReserved;
   long              dReserved;
   long              eReserved;
   long              fReserved;
   unsigned long     recordIDCount;
   unsigned long     attrTypeCount;
   Boolean           includeStartingPoint;
```

```
                                    /* if true, return the value specified
                                       by the starting indices */
   Byte              padByte;
   short             i1Reserved;
   Ptr               getBuffer;
   unsigned long     getBufferSize;
   unsigned long     startingRecordIndex;
                                    /* start from this record */
   unsigned long     startingAttrTypeIndex;
                                    /* start from this attribute type */
   Attribute         startingAttribute;
                                    /* start from this attribute value */
   long              pReserved;
};

typedef struct DirLookupGetPB DirLookupGetPB;

struct DirLookupParsePB {
   AuthDirParamHeader
   RecordIDPtr *           aRecordList;
                           /* must be same from the corresponding Get call */
   AttributeTypePtr *      attrTypeList;
                           /* must be same from the corresponding Get call */
   long                    cReserved;
   ForEachLookupRecordID   eachRecordID;
   ForEachAttrTypeLookup   eachAttrType;
   ForEachAttrValue        eachAttrValue;
   unsigned long           recordIDCount;
                           /* must be same from the corresponding Get call */
   unsigned long           attrTypeCount;
                           /* must be same from the corresponding Get call */
   long                    iReserved;
   Ptr                     getBuffer;
                           /* must be same from the corresponding Get call */
   unsigned long           getBufferSize;
                           /* must be same from the corresponding Get call */
   unsigned long           lastRecordIndex;
                           /* last record ID processed when parse
                              completed */
   unsigned long           lastAttributeIndex;
                           /* last attribute type processed when parse
                              completed */
   Attribute               lastAttribute;
                           /* last attribute value (with this CreationID)
```

```
                              processed when parse completed */
   unsigned long           attrSize;
                           /* length of the attribute that was not
                              returned */
};

typedef struct DirLookupParsePB DirLookupParsePB;

struct DirDeleteAttributeTypePB {
   AuthDirParamHeader
   RecordIDPtr       aRecord;
   AttributeTypePtr  attrType;
};

typedef struct DirDeleteAttributeTypePB DirDeleteAttributeTypePB;

struct DirEnumerateAttributeTypesGetPB {
   AuthDirParamHeader
   RecordIDPtr       aRecord;
   AttributeTypePtr  startingAttrType;
                                  /* starting point */
   long              cReserved;
   long              dReserved;
   long              eReserved;
   long              fReserved;
   long              gReserved;
   long              hReserved;
   Boolean           includeStartingPoint;
                                  /* if true, return the attrType
                                     specified by starting point */
   Byte              padByte;
   short             i1Reserved;
   Ptr               getBuffer;
   unsigned long     getBufferSize;
};

typedef struct DirEnumerateAttributeTypesGetPB
                                           DirEnumerateAttributeTypesGetPB;

struct DirEnumerateAttributeTypesParsePB {
   AuthDirParamHeader
   RecordIDPtr       aRecord;       /* Same as
                                     DirEnumerateAttributeTypesGetPB */
   long              bReserved;
   long              cReserved;
```

```
   long              dReserved;
   ForEachAttrType   eachAttrType;
   long              fReserved;
   long              gReserved;
   long              hReserved;
   long              iReserved;
   Ptr               getBuffer;
   unsigned long     getBufferSize;
};

typedef struct DirEnumerateAttributeTypesParsePB
                                          DirEnumerateAttributeTypesParsePB;

 struct DirGetDNodeAccessControlGetPB {
   AuthDirParamHeader
   PackedRLIPtr   pRLI;       /* RLI of the cluster whose access control
                                 list is sought */
   long           bReserved;  /* unused */
   long           cReserved;  /* unused */
   long           dReserved;  /* unused */
   long           eResreved;
   Boolean        forCurrentUserOnly;
   DSSpec         *startingPoint;
                              /* starting point */
   Boolean        includeStartingPoint;
                              /* if true, return the DsObject
                                 specified in starting point */
   Ptr            getBuffer;
   unsigned long  getBufferSize;
};

typedef struct DirGetDNodeAccessControlGetPB DirGetDNodeAccessControlGetPB;

struct DirGetDNodeAccessControlParsePB {
   AuthDirParamHeader
   PackedRLIPtr               pRLI;             /* RLI of the cluster  */
   long                       bReserved;        /* unused */
   long                       cReserved;        /* unused */
   long                       dReserved;        /* unused */
   ForEachDNodeAccessControl  eachObject;
   Boolean                    forCurrentUserOnly;
   DSSpec                     *startingPoint;   /* starting point */
   Boolean                    includeStartingPoint;
                                                /* if true, return the
```

```
                                                    record specified in
                                                    starting point */
   Ptr                          getBuffer;
   unsigned long                getBufferSize;

};

typedef struct DirGetDNodeAccessControlParsePB
                                          DirGetDNodeAccessControlParsePB;

struct DirGetRecordAccessControlGetPB {
   AuthDirParamHeader
   RecordIDPtr       aRecord;          /* ID of record whose access
                                          control list is sought */
   long              bReserved;        /* unused */
   long              cReserved;        /* unused */
   long              dReserved;        /* unused */
   long              eReserved;
   Boolean           forCurrentUserOnly;
   DSSpec            *startingPoint;   /* starting point */
   Boolean            includeStartingPoint;
                                       /* if true, return the DsObject
                                          specified in starting point */

   Ptr               getBuffer;
   unsigned long     getBufferSize;
};

typedef struct DirGetRecordAccessControlGetPB DirGetRecordAccessControlGetPB;

struct DirGetRecordAccessControlParsePB {
   AuthDirParamHeader
   RecordIDPtr                aRecord;    /* ID of record to which access
                                             control list is sought */
   long                       bReserved;  /* unused */
   long                       cReserved;  /* unused */
   long                       dReserved;  /* unused */
   ForEachRecordAccessControl eachObject;
   Boolean                    forCurrentUserOnly;
   DSSpec                     *startingPoint;
                                          /* starting point */
   Boolean                    includeStartingPoint;
                                          /* if true return the record
                                             specified in starting point */
```

```
   Ptr                        getBuffer;
   unsigned long              getBufferSize;
};

typedef struct DirGetRecordAccessControlParsePB
                                        DirGetRecordAccessControlParsePB;

struct DirGetAttributeAccessControlGetPB {
   AuthDirParamHeader
   RecordIDPtr       aRecord;       /* ID of record to which access control
                                       list is sought */
   AttributeTypePtr  aType;         /* attribute type to which access
                                       controls are sought */
   long              cReserved;     /* unused */
   long              dReserved;     /* unused */
   long              eResrved;
   Boolean           forCurrentUserOnly;

   DSSpec            *startingPoint;
                                    /* starting point */
   Boolean           includeStartingPoint;
                                    /* if true return the DsObject */
                                    /* specified in starting point */
   Ptr               getBuffer;
   unsigned longgetBufferSize;
};

typedef struct DirGetAttributeAccessControlGetPB
                                        DirGetAttributeAccessControlGetPB;

struct DirGetAttributeAccessControlParsePB {
   AuthDirParamHeader
   RecordIDPtr                  aRecord;    /* ID of record to which access
                                               control list is sought */
   AttributeTypePtr             aType;      /* attribute type to which
                                               access controls are sought */
   long                         cReserved;  /* unused */
   long                         dReserved;  /* unused */
   ForEachAttributeAccessControl eachObject;
   Boolean                      forCurrentUserOnly;
   DSSpec                       *startingPoint;
                                    /* starting point */
   Boolean                      includeStartingPoint;
                                 /* if true, return the record
```

```
                                                specified in starting point */
   Ptr                              getBuffer;
   unsigned long                    getBufferSize;
};

typedef struct DirGetAttributeAccessControlParsePB
                                        DirGetAttributeAccessControlParsePB;

struct DirAbortPB {
   AuthDirParamHeader
   union DirParamBlock  *pb;/* parameter block for the call that must
                               be aborted */
};

typedef struct DirAbortPB               DirAbortPB;

typedef union AuthParamBlock            AuthParamBlock;

typedef AuthParamBlock                  *AuthParamBlockPtr;

union DirParamBlock {
   struct {AuthDirParamHeader}          header;
   DirAddRecordPB                       addRecordPB;
   DirDeleteRecordPB                    deleteRecordPB;
   DirEnumerateGetPB                    enumerateGetPB;
   DirEnumerateParsePB                  enumerateParsePB;
   DirFindRecordGetPB                   findRecordGetPB;
   DirFindRecordParsePB                 findRecordParsePB;
   DirLookupGetPB                       lookupGetPB;
   DirLookupParsePB                     lookupParsePB;
   DirAddAttributeValuePB               addAttributeValuePB;
   DirDeleteAttributeTypePB             deleteAttributeTypePB;
   DirDeleteAttributeValuePB            deleteAttributeValuePB;
   DirChangeAttributeValuePB            changeAttributeValuePB;
   DirVerifyAttributeValuePB            verifyAttributeValuePB;
   DirFindValuePB                       findValuePB;
   DirEnumeratePseudonymGetPB           enumeratePseudonymGetPB;
   DirEnumeratePseudonymParsePB         enumeratePseudonymParsePB;
   DirAddPseudonymPB                    addPseudonymPB;
   DirDeletePseudonymPB                 deletePseudonymPB;
   DirAddAliasPB                        addAliasPB;
   DirEnumerateAttributeTypesGetPB      enumerateAttributeTypesGetPB;
   DirEnumerateAttributeTypesParsePB    enumerateAttributeTypesParsePB;
   DirGetNameAndTypePB                  getNameAndTypePB;
   DirSetNameAndTypePB                  setNameAndTypePB;
```

```
   DirGetRecordMetaInfoPB              getRecordMetaInfoPB;
   DirGetDNodeMetaInfoPB               getDNodeMetaInfoPB;
   DirGetDirectoryInfoPB               getDirectoryInfoPB;

   DirGetDNodeAccessControlGetPB       getDNodeAccessControlGetPB;
   DirGetDNodeAccessControlParsePB     getDNodeAccessControlParsePB;

   DirGetRecordAccessControlGetPB      getRecordAccessControlGetPB;
   DirGetRecordAccessControlParsePB    getRecordAccessControlParsePB;

   DirGetAttributeAccessControlGetPB   getAttributeAccessControlGetPB;
   DirGetAttributeAccessControlParsePB getAttributeAccessControlParsePB;

   DirEnumerateDirectoriesGetPB        enumerateDirectoriesGetPB;

   DirEnumerateDirectoriesParsePB      enumerateDirectoriesParsePB;
   DirAddADAPDirectoryPB               addADAPDirectoryPB;
   DirRemoveDirectoryPB                removeDirectoryPB;
   DirNetSearchADAPDirectoriesGetPB    netSearchADAPDirectoriesGetPB;
   DirNetSearchADAPDirectoriesParsePB  netSearchADAPDirectoriesParsePB;
   DirFindADAPDirectoryByNetSearchPB   findADAPDirectoryByNetSearchPB;
   DirMapDNodeNumberToPathNamePB       mapDNodeNumberToPathNamePB;
   DirMapPathNameToDNodeNumberPB       mapPathNameToDNodeNumberPB;
   DirGetLocalNetworkSpecPB            getLocalNetworkSpecPB;
   DirGetDNodeInfoPB                   getDNodeInfoPB;

/* calls for personal catalogs */
   DirCreatePersonalDirectoryPB        createPersonalDirectoryPB;
   DirOpenPersonalDirectoryPB          openPersonalDirectoryPB;
   DirClosePersonalDirectoryPB         closePersonalDirectoryPB;
   DirMakePersonalDirectoryRLIPB       makePersonalDirectoryRLIPB;

/* calls for CSAM's */

   DirAddDSAMPB                        addDSAMPB;
   DirInstantiateDSAMPB                instantiateDSAMPB;
   DirRemoveDSAMPB                     removeDSAMPB;
   DirAddDSAMDirectoryPB               addDSAMDirectoryPB;
   DirGetExtendedDirectoriesInfoPB     getExtendedDirectoriesInfoPB;
   DirGetDirectoryIconPB               getDirectoryIconPB;

/* call to dsRefNum for system(PowerTalk Setup) personal catalog */
   DirGetOCESetupRefNumPB              dirGetOCESetupRefNumPB;
```

```
/* abort a aysnchronous call */
   DirAbortPB                          abortPB;
};

typedef union DirParamBlock            DirParamBlock;
typedef DirParamBlock                  *DirParamBlockPtr;
```


Catalog Manager Functions

Getting Information About Catalogs

```
pascal OSErr DirEnumerateDirectoriesGet
                            (DirParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr DirEnumerateDirectoriesParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirFindRecordGet
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirFindRecordParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetDirectoryInfo
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetLocalNetworkSpec
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetDirectoryIcon
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetExtendedDirectoriesInfo
                            (DirParamBlockPtr paramBlock,Boolean async);
```

Getting Information About DNodes

```
pascal OSErr DirEnumerateGet
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirEnumerateParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetDNodeMetaInfo
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirMapDNodeNumberToPathName
                            (DirParamBlockPtrparamBlock,Boolean async);
pascal OSErr DirMapPathNameToDNodeNumber
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetDNodeInfo
                            (DirParamBlockPtr paramBlock,Boolean async);
```

Maintaining the PowerTalk Setup Catalog

```
pascal OSErr DirAddADAPDirectory
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirFindADAPDirectoryByNetSearch
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr irNetSearchADAPDirectoriesGet
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirNetSearchADAPDirectoriesParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirRemoveDirectory
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetOCESetupRefnum
                            (DirParamBlockPtr paramBlock,Boolean async);
```

Creating, Opening, and Closing Personal Catalogs

```
pascal OSErr DirCreatePersonalDirectory
                            (DirParamBlockPtr paramBlock);
pascal OSErr DirOpenPersonalDirectory
                            (DirParamBlockPtr paramBlock);
pascal OSErr DirClosePersonalDirectory
                            (DirParamBlockPtr paramBlock);
pascal OSErr DirMakePersonalDirectoryRLI
                            (DirParamBlockPtr paramBlock);
```

Managing Records

```
pascal OSErr DirAddRecord   (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirDeleteRecord
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetRecordMetaInfo
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetNameAndType
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirSetNameAndType
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirAddPseudonym
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirDeletePseudonym
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirEnumeratePseudonymGet
                            (DirParamBlockPtr paramBlock,Boolean async);
```

```
pascal OSErr DirEnumeratePseudonymParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirAddAlias    (DirParamBlockPtr paramBlock,Boolean async);
```

Managing Attribute Types and Values

```
pascal OSErr DirAddAttributeValue
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirDeleteAttributeValue
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirChangeAttributeValue
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirVerifyAttributeValue
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirFindValue   (DirParamBlockPtr paramBlock, Boolean async);
pascal OSErr DirLookupGet   (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirLookupParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirDeleteAttributeType
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirEnumerateAttributeTypesGet
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirEnumerateAttributeTypesParse
                            (DirParamBlockPtr paramBlock,Boolean async);
```

Reading Access Controls for dNodes, Records, and Attribute Types

```
pascal DSSpec *OCEGetAccessControlDSSpec
                            (const CategoryMask categoryBitMask);
pascal OSErr DirGetDNodeAccessControlGet
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetDNodeAccessControlParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetRecordAccessControlGet
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetRecordAccessControlParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetAttributeAccessControlGet
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetAttributeAccessControlParse
                            (DirParamBlockPtr paramBlock,Boolean async);
```

Canceling a Catalog Manager Function

```
pascal OSErr DirAbort       (DirParamBlockPtr paramBlock);
```

Application-Defined Functions

```
void MyCompletionRoutine    (DirParamBlockPtr paramBlk);
pascal Boolean MyForEachRecordID
                            (long clientData,const RecordID *recordID);
pascal Boolean MyForEachAttrType
                            (long clientData,
                             const AttributeType *attrType);
pascal Boolean MyForEachDirectory
                            (long clientData,
                             const DirectoryName *dirName,
                             const DirDiscriminator *discriminator,
                             DirGestalt features);
pascal Boolean MyForEachLookupRecordID
                            (long clientData,const RecordID *recordID);
pascal Boolean MyForEachAttrTypeLookup
                            (long clientData,
                             const AttributeType *attrType,
                             AccessMask myAttrAccMask);
pascal Boolean MyForEachAttrValue
                            (long clientData, const Attribute *attribute);
pascal Boolean MyForEachDirEnumSpec
                            (long clientData, const DirEnumSpec *enumSpec);
pascal Boolean MyForEachRecord
                            (long clientData,
                             const DirEnumSpec *enumSpec,
                             pRLI PackedRLIPtr);
pascal Boolean MyForEachADAPDirectory
                            (long clientData,
                             const DirectoryName *directoryName,
                             const DirDiscriminator *discriminator,
                             DirGestalt features, AddrBlock serverHint);
pascal Boolean MyForEachDNodeAccessControl
                            (long clientData, const DSSpec *dsObj,
                             AccessMask activeDnodeAccMask,
                             AccessMask defaultRecordAccMask,
                             AccessMask defaultAttributeAccMask);
```

```
pascal Boolean MyForEachRecordAccessControl
                            (long clientData,const DSSpec *dsObj,
                             AccessMask activeDnodeAccMask,
                             AccessMask activeRecordAccMask,
                             AccessMask defaultAttributeAccMask);
pascal Boolean MyForEachAttributeAccessControl
                            (long clientData,const DSSpec *dsObj,
                             AccessMask activeDnodeAccMask,
                             AccessMask activeRecordAccMask,
                             AccessMask activeAttributeAccMask);
```


Pascal Summary

Constants and Data Types

```
CONST
   {access categories bit numbers}
   kThisRecordOwnerBit           = 0;
   kFriendsBit                   = 1;
   kAuthenticatedInDNodeBit      = 2;
   kAuthenticatedInDirectoryBit  = 3;
   kGuestBit                     = 4;
   kMeBit                        = 5;

   {values of CategoryMask}
   kThisRecordOwnerMask          = $00000001;  {1<<kThisRecordOwnerBit}
   kFriendsMask                  = $00000002;  {1<<kFriendsBit}
   kAuthenticatedInDNodeMask     = $00000004;  {1<<kAuthenticatedInDNodeBit}
   kAuthenticatedInDirectoryMask = $00000008;
                                          {1<<kAuthenticatedInDirectoryBit}
   kGuestMask                    = $00000010;  {1<<kGuestBit}
   kMeMask                       = $00000020;  {1<<kMeBit}

   kEnumDistinguishedNameBit  = 0;
   kEnumAliasBit              = 1;
   kEnumPseudonymBit          = 2;
   kEnumDNodeBit              = 3;
   kEnumInvisibleBit          = 4;

   {values of DirEnumChoices}
   kEnumDistinguishedNameMask = $00000001;  {1<<kEnumDistinguishedNameBit}
   kEnumAliasMask             = $00000002;  {1<<kEnumAliasBit}
```

```
kEnumPseudonymMask          = $00000004;  {1<<kEnumPseudonymBit}
kEnumDNodeMask              = $00000008;  {1<<kEnumDNodeBit}
kEnumInvisibleMask          = $00000010;  {1<<kEnumInvisibleBit}

kEnumAllMask = (kEnumDistinguishedNameMask + kEnumAliasMask +
                kEnumPseudonymMask + kEnumDNodeMask + kEnumInvisibleMask);

{Values of DirSortOption}
kSortByName                                 = 0;
kSortByType                                 = 1;

{values of DirSortDirection}
kSortForwards                               = 0;
kSortBackwards                              = 1;

{values of DirMatchWith}
kMatchAll                                   = 0;
kExactMatch                                 = 1;
kBeginsWith                                 = 2;
kEndingWith                                 = 3;
kContaining                                 = 4;

kCurrentOCESortVersion                      = 1;

kSupportsDNodeNumberBit                     = 0;
kSupportsRecordCreationIDBit                = 1;
kSupportsAttributeCreationIDBit             = 2;
kSupportsMatchAllBit                        = 3;
kSupportsBeginsWithBit                      = 4;
kSupportsExactMatchBit                      = 5;
kSupportsEndsWithBit                        = 6;
kSupportsContainsBit                        = 7;
kSupportsOrderedEnumerationBit              = 8;
kCanSupportNameOrderBit                     = 9;
kCanSupportTypeOrderBit                     = 10;
kSupportSortBackwardsBit                    = 11;
kSupportIndexRatioBit                       = 12;
kSupportsEnumerationContinueBit             = 13;
kSupportsLookupContinueBit                  = 14;
kSupportsEnumerateAttributeTypeContinueBit  = 15;
kSupportsEnumeratePseudonymContinueBit      = 16;
kSupportsAliasesBit                         = 17;
kSupportsPseudonymsBit                      = 18;
kSupportsPartialPathNamesBit                = 19;
```

```
kSupportsAuthenticationBit                  = 20;
kSupportsProxiesBit                         = 21;
kSupportsFindRecordBit                      = 22;

{ values of DirGestalt }
kSupportsDNodeNumberMask                    = $00000001;
                            {1<<kSupportsDNodeNumberBit}
kSupportsRecordCreationIDMask               = $00000002;
                            {1<<kSupportsRecordCreationIDBit}
kSupportsAttributeCreationIDMask            = $00000004;
                            {1<<kSupportsAttributeCreationIDBit}
kSupportsMatchAllMask                       = $00000008;
                            {1<<kSupportsMatchAllBit}
kSupportsBeginsWithMask                     = $00000010;
                            {1<<kSupportsBeginsWithBit}
kSupportsExactMatchMask                     = $00000020;
                            {1<<kSupportsExactMatchBit}
kSupportsEndsWithMask                       = $00000040;
                            {1<<kSupportsEndsWithBit}
kSupportsContainsMask                       = $00000080;
                            {1<<kSupportsContainsBit}
kSupportsOrderedEnumerationMask             = $00000100;
                            {1<<kSupportsOrderedEnumerationBit}
kCanSupportNameOrderMask                    = $00000200;
                            {1<<kCanSupportNameOrderBit}
kCanSupportTypeOrderMask                    = $00000400;
                            {1<<kCanSupportTypeOrderBit}
kSupportSortBackwardsMask                   = $00000800;
                            {1<<kSupportSortBackwardsBit}
kSupportIndexRatioMask                      = $00001000;
                            {1<<kSupportIndexRatioBit}
kSupportsEnumerationContinueMask            = $00002000;
                            {1<<kSupportsEnumerationContinueBit}
kSupportsLookupContinueMask                 = $00004000;
                            {1<<kSupportsLookupContinueBit}
kSupportsEnumerateAttributeTypeContinueMask  = $00008000;
                            {1<<kSupportsEnumerateAttributeTypeContinueBit}
kSupportsEnumeratePseudonymContinueMask     = $00010000;
                            {1<<kSupportsEnumeratePseudonymContinueBit}
kSupportsAliasesMask                        = $00020000;
                            {1<<kSupportsAliasesBit}
kSupportsPseudonymsMask                     = $00040000;
                            {1<<kSupportsPseudonymsBit}
kSupportsPartialPathNamesMask               = $00080000;
```

```
                              {1<<kSupportsPartialPathNamesBit}
   kSupportsAuthenticationMask                  = $00100000;
                              {1<<kSupportsAuthenticationBit}
   kSupportsProxiesMask                         = $00200000;
                              {1<<kSupportsProxiesBit}
   kSupportsFindRecordMask                      = $00400000;
                              {1<<kSupportsFindRecordBit}

TYPE
   DirEnumChoices                   = LONGINT;
   DirMatchWith                     = BYTE;
   DirSortDirection                 = INTEGER;
   ForMyEachRecordID                = ProcPtr;
   ForMyEachLookupRecordID          = ProcPtr;
   ForMyEachAttrTypeLookup          = ProcPtr;
   ForMyEachAttrValue               = ProcPtr;
   ForMyEachAttrType                = ProcPtr;
   ForMyEachRecordID                = ProcPtr;
   ForMyEachDNodeAccessControl      = ProcPtr;
   ForMyEachRecordAccessControl     = ProcPtr;
   ForMyEachAttributeAccessControl  = ProcPtr;
   ForMyEachDirEnumSpec             = ProcPtr;
   ForMyEachDirectory               = ProcPtr;
   ForMyEachADAPDirectory           = ProcPtr;

DNodeID = RECORD
      dNodeNumber:   DNodeNum;      {dNode number}
      reserved1:     LONGINT;
      name:          RStringPtr;
      reserved2:     LONGINT;
   END;

   DirEnumSpec = RECORD
      enumFlag:      DirEnumChoices;
      indexRatio:    INTEGER;       {if supported, approx Record Position
                                     between 1 and 100; 0 If not supported}
      CASE INTEGER OF
         1: (recordIdentifier:   LocalRecordID);
         2: (dNodeIdentifier:    DNodeID);
   END;

   DirMetaInfo = RECORD
      info: ARRAY[1..4] OF LONGINT;
   END;
```

```
   SLRV = RECORD
      script:     ScriptCode;    {script code in which entries are sorted}
      language:   INTEGER;       {language code in which entries are sorted}
      regionCode: INTEGER;       {region code in which entries are sorted}
      version:    INTEGER;       {version of AOCE sorting software }
   END;

   AuthDirParamHeader = RECORD
      qLink:         Ptr;
      reserved1:     LONGINT;
      reserved2:     LONGINT;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LONGINT;
      reqCode:       INTEGER;
      reserved:      ARRAY[1..2] OF LONGINT;
      serverHint:    AddrBlock;
      dsRefNum:      INTEGER;
      callID:        LONGINT;
      identity:      AuthIdentity;
      gReserved1:    LONGINT;
      gReserved2:    LONGINT;
      gReserved3:    LONGINT;
      clientData:    LONGINT;
   END;

{Catalog types and operations}
   AuthIdentity         = LONGINT;        {unique identifier for an identity}
   LocalIdentity        = AuthIdentity;   {umbrella localIdentity}

   DirEnumerateDirectoriesGetPB = PACKED RECORD
      qLink:                    Ptr;
      reserved1:                LONGINT;
      reserved2:                LONGINT;
      ioCompletion:             ProcPtr;
      ioResult:                 OSErr;
      saveA5:                   LONGINT;
      reqCode:                  INTEGER;
      reserved:                 ARRAY[1..2] OF LONGINT;
      serverHint:               AddrBlock;
      dsRefNum:                 INTEGER;
      callID:                   LONGINT;
      identity:                 AuthIdentity;
      gReserved1:               LONGINT;
```

```
   gReserved2:                LONGINT;
   gReserved3:                LONGINT;
   clientData:                LONGINT;
   directoryKind:             OCEDirectoryKind; {enumerate catalogs
                                                 bearing this signature}
   startingDirectoryName:     DirectoryNamePtr; {starting catalog name}
   startingDirDiscriminator:  DirDiscriminator; {starting catalog
                                                 discriminator}
   eReserved:                 LONGINT;
   fReserved:                 LONGINT;
   gReserved:                 LONGINT;
   hReserved:                 LONGINT;
   includeStartingPoint:      BOOLEAN;          {if true, return catalog
                                                specified by starting
                                                point}
   padByte:                   Byte;
   i1Reserved:                INTEGER;
   getBuffer:                 Ptr;
   getBufferSize:             LONGINT;
END;

DirEnumerateDirectoriesParsePB = RECORD
   qLink:                     Ptr;
   reserved1:                 LONGINT;
   reserved2:                 LONGINT;
   ioCompletion:              ProcPtr;
   ioResult:                  OSErr;
   saveA5:                    LONGINT;
   reqCode:                   INTEGER;
   reserved:                  ARRAY[1..2] OF LONGINT;
   serverHint:                AddrBlock;
   dsRefNum:                  INTEGER;
   callID:                    LONGINT;
   identity:                  AuthIdentity;
   gReserved1:                LONGINT;
   gReserved2:                LONGINT;
   gReserved3:                LONGINT;
   clientData:                LONGINT;
   aReserved:                 LONGINT;
   bReserved:                 LONGINT;
   cReserved:                 LONGINT;
   dReserved:                 LONGINT;
   eachDirectory:             ForEachDirectory;
   fReserved:                 LONGINT;
```

```
   gReserved:                LONGINT;
   hReserved:                LONGINT;
   iReserved:                LONGINT;
   getBuffer:                Ptr;
   getBufferSize:            LONGINT;
END;

DirFindRecordGetPB = RECORD

   qLink:                    Ptr;
   reserved1:                LONGINT;
   reserved2:                LONGINT;
   ioCompletion:             ProcPtr;
   ioResult:                 OSErr;
   saveA5:                   LONGINT;
   reqCode:                  INTEGER;
   reserved:                 ARRAY[1..2] OF LONGINT;
   serverHint:               AddrBlock;
   dsRefNum:                 INTEGER;
   callID:                   LONGINT;
   identity:                 AuthIdentity;
   gReserved1:               LONGINT;
   gReserved2:               LONGINT;
   gReserved3:               LONGINT;
   clientData:               LONGINT;
   startingPoint:            RecordIDPtr;
   reservedA:                ARRAY[1..2] OF LONGINT;
   nameMatchString:          RStringPtr;
   typesList:                ^RStringPtr;
   typeCount:                LONGINT;
   reservedB:                LONGINT;
   reservedC:                INTEGER;
   matchNameHow:             DirMatchWith;
   matchTypeHow:             DirMatchWith;
   getBuffer:                Ptr;
   getBufferSize:            LONGINT;
   directoryName:            DirectoryNamePtr;
   discriminator:            DirDiscriminator;
END;

DirFindRecordParsePB = RECORD
   qLink:                    Ptr;
   reserved1:                LONGINT;
   reserved2:                LONGINT;
```

```
    ioCompletion:              ProcPtr;
    ioResult:                  OSErr;
    saveA5:                    LONGINT;
    reqCode:                   INTEGER;
    reserved:                  ARRAY[1..2] OF LONGINT;
    serverHint:                AddrBlock;
    dsRefNum:                  INTEGER;
    callID:                    LONGINT;
    identity:                  AuthIdentity;
    gReserved1:                LONGINT;
    gReserved2:                LONGINT;
    gReserved3:                LONGINT;
    clientData:                LONGINT;
    startingPoint:             RecordIDPtr;
    reservedA:                 ARRAY[1..2] OF LONGINT;
    nameMatchString:           RStringPtr;
    typesList:                 ^RStringPtr;
    typeCount:                 LONGINT;
    reservedB:                 LONGINT;
    reservedC:                 INTEGER;
    matchNameHow:              DirMatchWith;
    matchTypeHow:              DirMatchWith;
    getBuffer:                 Ptr;
    getBufferSize:             LONGINT;
    directoryName:             DirectoryNamePtr;
    discriminator:             DirDiscriminator;
    forEachRecordFunc:         ForEachRecord;
END;

DirGetDirectoryInfoPB = RECORD
    qLink:                     Ptr;
    reserved1:                 LONGINT;
    reserved2:                 LONGINT;
    ioCompletion:              ProcPtr;
    ioResult:                  OSErr;
    saveA5:                    LONGINT;
    reqCode:                   INTEGER;
    reserved:                  ARRAY[1..2] OF LONGINT;
    serverHint:                AddrBlock;
    dsRefNum:                  INTEGER;
    callID:                    LONGINT;
    identity:                  AuthIdentity;
    gReserved1:                LONGINT;
    gReserved2:                LONGINT;
```

```
      gReserved3:               LONGINT;
      clientData:               LONGINT;
      directoryName:            DirectoryNamePtr; {catalog name}
      discriminator:            DirDiscriminator; discriminate between
                                                  duplicate catalog
                                                  names}
      features:                 DirGestalt;       {capability bit flags}
   END;

DirGetLocalNetworkSpecPB = RECORD
      qLink:                    Ptr;
      reserved1:                LONGINT;
      reserved2:                LONGINT;
      ioCompletion:             ProcPtr;
      ioResult:                 OSErr;
      saveA5:                   LONGINT;
      reqCode:                  INTEGER;
      reserved:                 ARRAY[1..2] OF LONGINT;
      serverHint:               AddrBlock;
      dsRefNum:                 INTEGER;
      callID:                   LONGINT;
      identity:                 AuthIdentity;
      gReserved1:               LONGINT;
      gReserved2:               LONGINT;
      gReserved3:               LONGINT;
      clientData:               LONGINT;
      directoryName:            DirectoryNamePtr;    {catalog name}
      discriminator:            DirDiscriminator;    {discriminator}
      networkSpec:              NetworkSpecPtr;      {NetworkSpec}

   END;

DirGetDirectoryIconPB = RECORD
      qLink:                    Ptr;
      reserved1:                LONGINT;
      reserved2:                LONGINT;
      ioCompletion:             ProcPtr;
      ioResult:                 OSErr;
      saveA5:                   LONGINT;
      reqCode:                  INTEGER;
      reserved:                 ARRAY[1..2] OF LONGINT;
      serverHint:               AddrBlock;
      dsRefNum:                 INTEGER;
      callID:                   LONGINT;
```

```
      identity:                 AuthIdentity;
      gReserved1:               LONGINT;
      gReserved2:               LONGINT;
      gReserved3:               LONGINT;
      clientData:               LONGINT;
      pRLI:                     PackedRLIPtr;  {packed RLI for the catalog}
      iconType:                 OSType;        {type of Icon requested}
      iconBuffer:               Ptr;           {buffer to hold Icon Data}
      bufferSize:               LONGINT;       {size of buffer to hold icon
                                                data}
      END;

DirGetExtendedDirectoriesInfoPB = RECORD
      qLink:                    Ptr;
      reserved1:                LONGINT;
      reserved2:                LONGINT;
      ioCompletion:             ProcPtr;
      ioResult:                 OSErr;
      saveA5:                   LONGINT;
      reqCode:                  INTEGER;
      reserved:                 ARRAY[1..2] OF LONGINT;
      serverHint:               AddrBlock;
      dsRefNum:                 INTEGER;
      callID:                   LONGINT;
      identity:                 AuthIdentity;
      gReserved1:               LONGINT;
      gReserved2:               LONGINT;
      gReserved3:               LONGINT;
      clientData:               LONGINT;
      buffer:                   Ptr;           {Pointer to a buufer
                                                where data is returned}
      bufferSize:               LONGINT;       {Length of buffer in which
                                               actual data is returned}
      totalEntries:             LONGINT;    {total number of catalogs found}
      actualEntries:            LONGINT;    {total number of catalog
                                             entries returned}
      END;

   ForEachDirectory = ProcPtr;
      {FUNCTION ForEachDirectory(clientData: long; dirName: DirectoryNamePtr;
      discriminator: DirDiscriminator; features: DirGestalt): BOOLEAN;}
```

```
DirEnumerateGetPB = PACKED RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   aRLI:                PackedRLIPtr;     {an RLI specifying the cluster
                                          to be enumerated}
   startingPoint:       ^DirEnumSpec;
   sortBy:              DirSortOption;
   sortDirection:       DirSortDirection;
   dReserved:           LONGINT;
   nameMatchString:     RStringPtr;       {name from which enumeration
                                          should start}
   typesList:           ^RStringPtr;      {list of entity types to be
                                          enumerated}
   typeCount:           LONGINT;          {number of types in the list}
   enumFlags:           DirEnumChoices;   {indicates what to enumerate}
   includeStartingPoint:BOOLEAN;          {if true return the record
                                           specified in starting point}
   padByte:             Byte;
   matchNameHow:        DirMatchWith;     {matching Criteria}
                                          {for nameMatchString}
   matchTypeHow:        DirMatchWith;     {matching criteria for typeList}
   getBuffer:           Ptr;
   getBufferSize:       LONGINT;
   responseSLRV:        SLRV;             {response SLRV}
END;

DirEnumerateParsePB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
```

```
    ioCompletion:  ProcPtr;
    ioResult:      OSErr;
    saveA5:        LONGINT;
    reqCode:       INTEGER;
    reserved:      ARRAY[1..2] OF LONGINT;
    serverHint:    AddrBlock;
    dsRefNum:      INTEGER;
    callID:        LONGINT;
    identity:      AuthIdentity;
    gReserved1:    LONGINT;
    gReserved2:    LONGINT;
    gReserved3:    LONGINT;
    clientData:    LONGINT;
    aRLI:          PackedRLIPtr;           {an RLI specifying the cluster
                                            to be enumerated}
    bReserved:     LONGINT;
    cReserved:     LONGINT;
    eachEnumSpec:  ForEachDirEnumSpec;
    eReserved:     LONGINT;
    fReserved:     LONGINT;
    gReserved:     LONGINT;
    hReserved:     LONGINT;
    iReserved:     LONGINT;
    getBuffer:     Ptr;
    getBufferSize: LONGINT;
    l1Reserved:    INTEGER;
    l2Reserved:    INTEGER;
    l3Reserved:    INTEGER;
    l4Reserved:    INTEGER;
END;

DirGetDNodeMetaInfoPB = RECORD
    qLink:         Ptr;
    reserved1:     LONGINT;
    reserved2:     LONGINT;
    ioCompletion:  ProcPtr;
    ioResult:      OSErr;
    saveA5:        LONGINT;
    reqCode:       INTEGER;
    reserved:      ARRAY[1..2] OF LONGINT;
    serverHint:    AddrBlock;
    dsRefNum:      INTEGER;
    callID:        LONGINT;
    identity:      AuthIdentity;
```

```
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   pRLI:          PackedRLIPtr;
   metaInfo:      DirMetaInfo;
END;

DirMapDNodeNumberToPathNamePB = RECORD
   qLink:            Ptr;
   reserved1:        LONGINT;
   reserved2:        LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   reserved:         ARRAY[1..2] OF LONGINT;
   serverHint:       AddrBlock;
   dsRefNum:         INTEGER;
   callID:           LONGINT;
   identity:         AuthIdentity;
   gReserved1:       LONGINT;
   gReserved2:       LONGINT;
   gReserved3:       LONGINT;
   clientData:       LONGINT;
   directoryName:    DirectoryNamePtr;       {catalog name}
   discriminator:    DirDiscriminator;       {discriminator}
   dNodeNumber:      DNodeNum;               {dNodenumber to be mapped}
   path:             PackedPathNamePtr;      {packed path name returned}
   lengthOfPathName: INTEGER;                {length of packed pathname
                                              structure}
END;

DirMapPathNameToDNodeNumberPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
```

```
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   directoryName: DirectoryNamePtr;          {catalog name}
   discriminator: DirDiscriminator;          {discriminator}
   dNodeNumber:   DNodeNum;                  {dNode number to the path}
   path:          PackedPathNamePtr;         {pathname to be mapped}
END;

DirGetDNodeInfoPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   pRLI:          PackedRLIPtr;     {packed RLI whose info is requested}
   descriptor:    DirNodeKind;      {dNode descriptor}
   networkSpec:   NetworkSpecPtr;   {cluster's networkSpec if kIsCluster}
END;

DirAddADAPDirectoryPB = PACKED RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
```

```
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   directoryName:       DirectoryNamePtr;    {catalog name}
   discriminator:       DirDiscriminator;    {discriminate between}
                                             duplicate catalog names}
   addToOCESetup:       BOOLEAN;             {add this catalog to
                                              PowerTalk setup}
   padByte:             Byte;
   directoryRecordCID:  CreationID;          {creation ID for the
                                             catalog record}
END;

DirFindADAPDirectoryByNetSearchPB = PACKED RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   directoryName:       DirectoryNamePtr; {catalog name}
   discriminator:       DirDiscriminator; {discriminate between
                                           duplicate names}
   addToOCESetup:       BOOLEAN;          {add this catalog to PowerTalk
                                          Setup list}
   padByte:             Byte;
   directoryRecordCID:  CreationID;       {creation ID for the catalog
                                           record}
END;
```

```
   DirNetSearchADAPDirectoriesGetPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   getBuffer:     Ptr;
   getBufferSize: LONGINT;
   cReserved:     LONGINT;
END;

DirNetSearchADAPDirectoriesParsePB = RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   getBuffer:           Ptr;
   getBufferSize:       LONGINT;
   eachADAPDirectory:   ForEachADAPDirectory;
END;
```

```
ForEachADAPDirectory = ProcPtr;
   {FUNCTION ForEachADAPDirectory(
      clientData: long;
      dirName: DirectoryNamePtr;
      discriminator: DirDiscriminator;
      features: DirGestalt;
      serverHint: AddrBlock): BOOLEAN;}

DirRemoveDirectoryPB = RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   directoryRecordCID:  CreationID; {creation ID for the catalog record}
END;

DirRemoveDirectoryPB = RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
```

```
   clientData:          LONGINT;
   directoryRecordCID:  CreationID; {creation ID for the catalog record}
END;

DirGetOCESetupRefNumPB = RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   oceSetupRecordCID:   CreationID; {creation ID for the catalog record}
END;

DirCreatePersonalDirectoryPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   fsSpec:        FSSpecPtr;     {FSSpec for the personal catalog}
```

```
   fdType:        OSType;        {file type for the personal catalog}
   fdCreator:     OSType;        {file creator for the personal catalog}
END;

DirOpenPersonalDirectoryPB = PACKED RECORD
   qLink:            Ptr;
   reserved1:        LONGINT;
   reserved2:        LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   reserved:         ARRAY[1..2] OF LONGINT;
   serverHint:       AddrBlock;
   dsRefNum:         INTEGER;
   callID:           LONGINT;
   identity:         AuthIdentity;
   gReserved1:       LONGINT;
   gReserved2:       LONGINT;
   gReserved3:       LONGINT;
   clientData:       LONGINT;
   fsSpec:           FSSpecPtr;     {open an existing personal catalog}
   accessRequested:  Char;          {open: permissions requested (byte)}
   accessGranted:    Char;          {open: permissions (byte) (Granted)
   features:         DirGestalt;    {features for personal catalog}
END;

DirClosePersonalDirectoryPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
```

```
   gReserved3:    LONGINT;
   clientData:    LONGINT;
END;

DirMakePersonalDirectoryRLIPB = RECORD
   qLink:            Ptr;
   reserved1:        LONGINT;
   reserved2:        LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   reserved:         ARRAY[1..2] OF LONGINT;
   serverHint:       AddrBlock;
   dsRefNum:         INTEGER;
   callID:           LONGINT;
   identity:         AuthIdentity;
   gReserved1:       LONGINT;
   gReserved2:       LONGINT;
   gReserved3:       LONGINT;
   clientData:       LONGINT;
   fromFSSpec:       FSSpecPtr;     {FSSpec for creating relative alias}
   pRLIBufferSize:   INTEGER;       {length of 'pRLI' buffer}
   pRLISize:         INTEGER;       {length of actual 'pRLI'}
   pRLI:             PackedRLIPtr;  {pRLI for the specified address book}
END;

DirAddRecordPB = RECORD
   qLink:            Ptr;
   reserved1:        LONGINT;
   reserved2:        LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   reserved:         ARRAY[1..2] OF LONGINT;
   serverHint:       AddrBlock;
   dsRefNum:         INTEGER;
   callID:           LONGINT;
   identity:         AuthIdentity;
   gReserved1:       LONGINT;
   gReserved2:       LONGINT;
   gReserved3:       LONGINT;
   clientData:       LONGINT;
```

```
   aRecord:          RecordIDPtr;            {creation ID returned here}
   allowDuplicate:   BOOLEAN;
END;

DirDeleteRecordPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   aRecord:       RecordIDPtr;
END;

DirGetRecordMetaInfoPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   aRecord:       RecordIDPtr;
   metaInfo:      DirMetaInfo;
END;
```

```
DirGetNameAndTypePB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    Longint;
   aRecord:       RecordIDPtr;
END;

DirSetNameAndTypePB = PACKED RECORD
   qLink:            Ptr;
   reserved1:        LONGINT;
   reserved2:        LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   reserved:         ARRAY[1..2] OF LONGINT;
   serverHint:       AddrBlock;
   dsRefNum:         INTEGER;
   callID:           LONGINT;
   identity:         AuthIdentity;
   gReserved1:       LONGINT;
   gReserved2:       LONGINT;
   gReserved3:       LONGINT;
   clientData:       LONGINT;
   aRecord:          RecordIDPtr;
   allowDuplicate:   BOOLEAN;
   padByte:          Byte;
   newName:          RStringPtr;            {new name for the record}
   newType:          RStringPtr;            {new type for the record}
END;
```

```
DirAddPseudonymPB = RECORD
   qLink:            Ptr;
   reserved1:        LONGINT;
   reserved2:        LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   reserved:         ARRAY[1..2] OF LONGINT;
   serverHint:       AddrBlock;
   dsRefNum:         INTEGER;
   callID:           LONGINT;
   identity:         AuthIdentity;
   gReserved1:       LONGINT;
   gReserved2:       LONGINT;
   gReserved3:       LONGINT;
   clientData:       LONGINT;
   aRecord:          RecordIDPtr;   {record ID to be added to pseudonym}
   pseudonymName:    RStringPtr;    {new name to be added as pseudonym}
   pseudonymType:    RStringPtr;    {new name to be added as pseudonym}
   allowDuplicate:   BOOLEAN;
END;

DirDeletePseudonymPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   aRecord:       RecordIDPtr;         {record ID to which pseudonym is
                                        to be added}
```

```
   pseudonymName: RStringPtr;          {pseudonymName to be deleted}
   pseudonymType: RStringPtr;          {pseudonymType to be deleted}
END;

DirEnumeratePseudonymGetPB = PACKED RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   aRecord:                RecordIDPtr;
   startingName:           RStringPtr;
   startingType:           RStringPtr;
   dReserved:              LONGINT;
   eReserved:              LONGINT;
   fReserved:              LONGINT;
   gReserved:              LONGINT;
   hReserved:              LONGINT;
   includeStartingPoint:   BOOLEAN; {if true, the pseudonym specified}
                                     by starting point will be included}
   padByte:                Byte;
   i1Reserved:             INTEGER;
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;

DirEnumeratePseudonymParsePB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
```

```
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   aRecord:       RecordIDPtr;      {same as DirEnumerateAliasesGetPB}
   bReserved:     LONGINT;
   cReserved:     LONGINT;
   eachRecordID:  ForEachRecordID;
   eReserved:     LONGINT;
   fReserved:     LONGINT;
   gReserved:     LONGINT;
   hReserved:     LONGINT;
   iReserved:     LONGINT;
   getBuffer:     Ptr;
   getBufferSize: LONGINT;
END;

DirAddAliasPB = RECORD
   qLink:            Ptr;
   reserved1:        LONGINT;
   reserved2:        LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   reserved:         ARRAY[1..2] OF LONGINT;
   serverHint:       AddrBlock;
   dsRefNum:         INTEGER;
   callID:           LONGINT;
   identity:         AuthIdentity;
   gReserved1:       LONGINT;
   gReserved2:       LONGINT;
   gReserved3:       LONGINT;
   clientData:       LONGINT;
   aRecord:          RecordIDPtr;
   allowDuplicate:   BOOLEAN;
END;
DirAddAttributeValuePB = RECORD
```

```
    qLink:          Ptr;
    reserved1:      LONGINT;
    reserved2:      LONGINT;
    ioCompletion:   ProcPtr;
    ioResult:       OSErr;
    saveA5:         LONGINT;
    reqCode:        INTEGER;
    reserved:       ARRAY[1..2] OF LONGINT;
    serverHint:     AddrBlock;
    dsRefNum:       INTEGER;
    callID:         LONGINT;
    identity:       AuthIdentity;
    gReserved1:     LONGINT;
    gReserved2:     LONGINT;
    gReserved3:     LONGINT;
    clientData:     Longint;
    aRecord:        RecordIDPtr;
    attr:           AttributePtr;
END;

DirDeleteAttributeValuePB = RECORD
    qLink:          Ptr;
    reserved1:      LONGINT;
    reserved2:      LONGINT;
    ioCompletion:   ProcPtr;
    ioResult:       OSErr;
    saveA5:         LONGINT;
    reqCode:        INTEGER;
    reserved:       ARRAY[1..2] OF LONGINT;
    serverHint:     AddrBlock;
    dsRefNum:       INTEGER;
    callID:         LONGINT;
    identity:       AuthIdentity;
    gReserved1:     LONGINT;
    gReserved2:     LONGINT;
    gReserved3:     LONGINT;
    clientData:     LONGINT;
    aRecord:        RecordIDPtr;
    attr:           AttributePtr;
END;

DirChangeAttributeValuePB = RECORD
    qLink:          Ptr;
    reserved1:      LONGINT;
```

```
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   aRecord:       RecordIDPtr;
   currentAttr:   AttributePtr;
   newAttr:       AttributePtr;
END;

DirVerifyAttributeValuePB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   aRecord:       RecordIDPtr;
   attr:          AttributePtr;
END;

DirFindValuePB = RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
```

```
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   aRLI:                PackedRLIPtr;     {an RLI specifying the cluster
                                           to be enumerated}
   aRecord:             LocalRecordIDPtr; {if not nil, look only in this
                                           record}
   attrType:            AttributeTypePtr; {if not nil, look only in this
                                           attribute type}
   startingRecord:      LocalRecordIDPtr; {record in which to start
                                           searching}
   startingAttribute:   AttributePtr;     {attribute in which to start
                                           searching}
   recordFound:         LocalRecordIDPtr; {record in which data was found}
   attributeFound:      Attribute;        {attribute in which data was
                                           found}
   matchSize:           LONGINT;          {length of matching bytes}
   matchingData:        Ptr;              {data bytes to be matched in
                                           search}
   sortDirection:       DirSortDirection; {sort direction (forwards or
                                           backwards)}
END;

DirLookupGetPB = RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
```

```
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   aRecordList:            ^RecordIDPtr;     {an array of record ID
                                              pointers}
   attrTypeList:           ^AttributeTypePtr;{an array of attribute types}
   cReserved:              LONGINT;
   dReserved:              LONGINT;
   eReserved:              LONGINT;
   fReserved:              LONGINT;
   recordIDCount:          LONGINT;
   attrTypeCount:          LONGINT;
   includeStartingPoint:   BOOLEAN;          {if true, return the value
                                              specified by the starting
                                              indices}
   {padByte:               Byte;}
   i1Reserved:             INTEGER;
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
   startingRecordIndex:    LONGINT;          {start from this record}
   startingAttrTypeIndex:  LONGINT;          {start from this attribute
                                              type}
   startingAttribute:      Attribute;        {start from this attribute
                                              value}
   pReserved:              LONGINT;
END;

DirLookupParsePB = RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
```

```
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   aRecordList:         ^RecordIDPtr;     {must be same from the
                                           corresponding Get call}
   attrTypeList:        ^AttributeTypePtr;{must be same from the
                                           corresponding Get call}
   cReserved:           LONGINT;
   eachRecordID:        ForEachLookupRecordID;
   eachAttrType:        ForEachAttrTypeLookup;
   eachAttrValue:       ForEachAttrValue;
   recordIDCount:       LONGINT;          {must be same from the
                                           corresponding Get call}
   attrTypeCount:       LONGINT;          {must be same from the
                                           corresponding Get call}
   iReserved:           LONGINT;
   getBuffer:           Ptr;              {must be same from the
                                           corresponding Get call}
   getBufferSize:       LONGINT;          {must be same from the
                                           corresponding Get call}
   lastRecordIndex:     LONGINT;          {last RecordID processed when
                                           parse completed}
   lastAttributeIndex:  LONGINT;          {last Attribute Type processed
                                           when parse completed}
   lastAttribute:       Attribute;        {last attribute value (with
                                           this creation ID) processed
                                           when parse completed}
   attrSize:            LONGINT;          {length of the attribute that
                                           was not returned}
END;

DirDeleteAttributeTypePB = RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
```

```
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   aRecord:             RecordIDPtr;
   attrType:            AttributeTypePtr;
END;

DirEnumerateAttributeTypesGetPB = PACKED RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   aRecord:                RecordIDPtr;
   startingAttrType:       AttributeTypePtr;{starting point}
   cReserved:              LONGINT;
   dReserved:              LONGINT;
   eReserved:              LONGINT;
   fReserved:              LONGINT;
   gReserved:              LONGINT;
   hReserved:              LONGINT;
   includeStartingPoint:   BOOLEAN;    {if true, return the attribute
                                        Type specified by starting point}
   padByte:                Byte;
   i1Reserved:             INTEGER;
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;

DirEnumerateAttributeTypesParsePB = RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
```

```
    reserved2:              LONGINT;
    ioCompletion:           ProcPtr;
    ioResult:               OSErr;
    saveA5:                 LONGINT;
    reqCode:                INTEGER;
    reserved:               ARRAY[1..2] OF LONGINT;
    serverHint:             AddrBlock;
    dsRefNum:               INTEGER;
    callID:                 LONGINT;
    identity:               AuthIdentity;
    gReserved1:             LONGINT;
    gReserved2:             LONGINT;
    gReserved3:             LONGINT;
    clientData:             LONGINT;
    aRecord:                RecordIDPtr;{Same as
                                        DirEnumerateAttributeTypesGetPB}
    bReserved:              LONGINT;
    cReserved:              LONGINT;
    dReserved:              LONGINT;
    eachAttrType:           ForEachAttrType;
    fReserved:              LONGINT;
    gReserved:              LONGINT;
    hReserved:              LONGINT;
    iReserved:              LONGINT;
    getBuffer:              Ptr;
    getBufferSize:          LONGINT;
END;

DirGetDNodeAccessControlGetPB = RECORD
    qLink:                  Ptr;
    reserved1:              LONGINT;
    reserved2:              LONGINT;
    ioCompletion:           ProcPtr;
    ioResult:               OSErr;
    saveA5:                 LONGINT;
    reqCode:                INTEGER;
    reserved:               ARRAY[1..2] OF LONGINT;
    serverHint:             AddrBlock;
    dsRefNum:               INTEGER;
    callID:                 LONGINT;
    identity:               AuthIdentity;
    gReserved1:             LONGINT;
    gReserved2:             LONGINT;
    gReserved3:             LONGINT;
```

```
   clientData:             LONGINT;
   pRLI:                   PackedRLIPtr;  {RLI of the cluster whose
                                           access control list is sought}
   bReserved:              LONGINT;
   cReserved:              LONGINT;
   dReserved:              LONGINT;
   eResreved:              LONGINT;
   forCurrentUserOnly:     BOOLEAN;
   startingPoint:          ^DSSpec;       {starting point}
   includeStartingPoint:   BOOLEAN;       {if true, return the DsObject
                                           specified in starting point}
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;

DirGetDNodeAccessControlParsePB = RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   pRLI:                   PackedRLIPtr;              {RLI of the cluster}
   bReserved:              LONGINT;                   {unused}
   cReserved:              LONGINT;                   {unused}
   dReserved:              LONGINT;                   {unused}
   eachObject:             ForEachDNodeAccessControl;
   forCurrentUserOnly:     BOOLEAN;
   startingPoint:          ^DSSpec;                   {starting point}
   includeStartingPoint:   BOOLEAN;                   {if true, return
                                                       the record
                                                       specified in
                                                       in starting point}
```

```
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;

DirGetRecordAccessControlGetPB = RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   aRecord:                RecordIDPtr;   {RecordID whose access
                                           control list is sought }
   bReserved:              LONGINT;       {unused}
   cReserved:              LONGINT;       {unused}
   dReserved:              LONGINT;       {unused}
   eResreved:              LONGINT;
   forCurrentUserOnly:     BOOLEAN;
   startingPoint:          ^DSSpec;       {starting Point}
   includeStartingPoint:   BOOLEAN;       {if true, return the DsObject
                                           specified in starting point}
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;

DirGetRecordAccessControlParsePB = RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
```

```
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   aRecord:                RecordIDPtr;         {RecordID whose access
                                                 control list is sought}
   bReserved:              LONGINT;             {unused}
   cReserved:              LONGINT;             {unused}
   dReserved:              LONGINT;             {unused}
   eachObject:             ForEachRecordAccessControl;
   forCurrentUserOnly:     BOOLEAN;
   startingPoint:          ^DSSpec;             {starting point}
   includeStartingPoint:   BOOLEAN;             {if true, return the
                                                 record specified in}
                                                {starting point}
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;

DirGetAttributeAccessControlGetPB = RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   aRecord:                RecordIDPtr;      {RecordID whose access
                                              control list is sought}
   aType:                  AttributeTypePtr; {attribute type to which
                                              access controls are sought}
```

```
   cReserved:              LONGINT;
   dReserved:              LONGINT;          {unused}
   eResreved:              LONGINT;
   forCurrentUserOnly:     BOOLEAN;
   includeStartingPoint:   BOOLEAN;          {if true, return the DsObject
                                              specified in starting point}
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;

DirGetAttributeAccessControlParsePB = RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   aRecord:                RecordIDPtr;      {record ID whose access
                                              control list is sought}
   aType:                  AttributeTypePtr; {attribute type whose
                                              access controls are sought}
   cReserved:              LONGINT;
   dReserved:              LONGINT;
   eachObject:             ForEachAttributeAccessControl;
   forCurrentUserOnly:     BOOLEAN;
   startingPoint:          ^DSSpec;          {starting Point }
   includeStartingPoint:   BOOLEAN;          {if true, return the record
                                              specified in starting point}
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;
```

```
DirAbortPB = RECORD
   qLink: Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   pb:            Ptr;        {parameter block for the call that must be
                               aborted {^DirParamBlock}}
END;

DirParamBlock = RECORD
   CASE INTEGER OF
   1: (header:                      AuthDirParamHeader);
   2: (addRecordPB:                 DirAddRecordPB);
   3: (deleteRecordPB:              DirDeleteRecordPB);
   4: (enumerateGetPB:              DirEnumerateGetPB);
   5: (enumerateParsePB:            DirEnumerateParsePB);
   6: (findRecordGetPB:             DirFindRecordGetPB);
   7: (findRecordParsePB:           DirFindRecordParsePB);
   8:(lookupGetPB:                  DirLookupGetPB);
   9:(lookupParsePB:                DirLookupParsePB);
   10:(addAttributeValuePB:         DirAddAttributeValuePB);
   11:(deleteAttributeTypePB:       DirDeleteAttributeTypePB);
   12:(deleteAttributeValuePB:      DirDeleteAttributeValuePB);
   13:(changeAttributeValuePB:      DirChangeAttributeValuePB);
   14:(verifyAttributeValuePB:      DirVerifyAttributeValuePB);
   15:(findValuePB:                 DirFindValuePB);
   16:(enumeratePseudonymGetPB:     DirEnumeratePseudonymGetPB);
   17:(enumeratePseudonymParsePB:   DirEnumeratePseudonymParsePB);
   18:(addPseudonymPB:              DirAddPseudonymPB);
   19:(deletePseudonymPB:           DirDeletePseudonymPB);
   20:(addAliasPB:                  DirAddAliasPB);
   21:(enumerateAttributeTypesGetPB:DirEnumerateAttributeTypesGetPB);
```

```
22:(enumerateAttributeTypesParsePB:
                                DirEnumerateAttributeTypesParsePB);
23:(getNameAndTypePB:           DirGetNameAndTypePB);
24:(setNameAndTypePB:           DirSetNameAndTypePB);
25:(getRecordMetaInfoPB:        DirGetRecordMetaInfoPB);
26:(getDNodeMetaInfoPB:         DirGetDNodeMetaInfoPB);
27:(getDirectoryInfoPB:         DirGetDirectoryInfoPB);
28:(getDNodeAccessControlGetPB:  DirGetDNodeAccessControlGetPB);
29:(getDNodeAccessControlParsePB:DirGetDNodeAccessControlParsePB);
30:(getRecordAccessControlGetPB: DirGetRecordAccessControlGetPB);
31:(getRecordAccessControlParsePB:
                                DirGetRecordAccessControlParsePB);
32:(getAttributeAccessControlGetPB:
                                DirGetAttributeAccessControlGetPB);
33:(getAttributeAccessControlParsePB:
                                DirGetAttributeAccessControlParsePB);
34: (enumerateDirectoriesGetPB:  DirEnumerateDirectoriesGetPB);
35: (enumerateDirectoriesParsePB: DirEnumerateDirectoriesParsePB);
36: (addADAPDirectoryPB:         DirAddADAPDirectoryPB);
37: (removeDirectoryPB:          DirRemoveDirectoryPB);
38: (netSearchADAPDirectoriesGetPB:
                                DirNetSearchADAPDirectoriesGetPB);
39: (netSearchADAPDirectoriesParsePB:
                                DirNetSearchADAPDirectoriesParsePB);
40: (findADAPDirectoryByNetSearchPB:
                                DirFindADAPDirectoryByNetSearchPB);
41: (mapDNodeNumberToPathNamePB: DirMapDNodeNumberToPathNamePB);
42: (mapPathNameToDNodeNumberPB: DirMapPathNameToDNodeNumberPB);
43: (getLocalNetworkSpecPB:      DirGetLocalNetworkSpecPB);
44: (getDNodeInfoPB:             DirGetDNodeInfoPB);

{calls for personal catalogs}

45: (createPersonalDirectoryPB:  DirCreatePersonalDirectoryPB);
46: (openPersonalDirectoryPB:    DirOpenPersonalDirectoryPB);
47: (closePersonalDirectoryPB:   DirClosePersonalDirectoryPB);
48: (makePersonalDirectoryRLIPB: DirMakePersonalDirectoryRLIPB);

{calls For CSAMs}

49: (addDSAMPB:                  DirAddDSAMPB);
50: (instantiateDSAMPB:          DirInstantiateDSAMPB);
51: (removeDSAMPB:               DirRemoveDSAMPB);
52: (addDSAMDirectoryPB:         DirAddDSAMDirectoryPB);
```

```
      53: (getExtendedDirectoriesInfoPB:
                                    DirGetExtendedDirectoriesInfoPB);
      54: (getDirectoryIconPB:      DirGetDirectoryIconPB);

      {call to dsRefNum for system(Setup: PowerTalk) personal catalog}

      55: (dirGetOCESetupRefNumPB:  DirGetOCESetupRefNumPB);

      {abort a aysnchronous call}

      56: (abortPB:                 DirAbortPB);

   END;

   DirParamBlockPtr = ^DirParamBlock;
```


Catalog Manager Functions

Getting Information About Catalogs

```
FUNCTION DirEnumerateDirectoriesGet
                           (paramBlock: DirParamBlockPtr;
                            async: BOOLEAN): OSErr;
FUNCTION DirEnumerateDirectoriesParse
                           (paramBlock: DirParamBlockPtr;
                            async: BOOLEAN): OSErr;
FUNCTION DirFindRecordGet  (paramBlock: DirParamBlockPtr;
                            async: BOOLEAN): OSErr;
FUNCTION DirFindRecordParse (paramBlock: DirParamBlockPtr;
                            async: BOOLEAN): OSErr;
FUNCTION DirGetDirectoryInfo
                           (paramBlock: DirParamBlockPtr;
                            async: BOOLEAN): OSErr;
FUNCTION DirGetLocalNetworkSpec
                           (paramBlock: DirParamBlockPtr;
                            async: BOOLEAN): OSErr;
FUNCTION DirGetDirectoryIcon
                           (paramBlock: DirParamBlockPtr;
                            async: BOOLEAN): OSErr;
FUNCTION DirGetExtendedDirectoriesInfo
                           (paramBlock: DirParamBlockPtr;
                            async: BOOLEAN): OSErr;
```

Getting Information About DNodes

```
FUNCTION DirEnumerateGet    (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirEnumerateParse  (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetDNodeMetaInfo
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirMapDNodeNumberToPathName
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirMapPathNameToDNodeNumber
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetDNodeInfo    (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
```

Maintaining the PowerTalk Setup Catalog

```
FUNCTION DirAddADAPDirectory
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirFindADAPDirectoryByNetSearch
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirNetSearchADAPDirectoriesGet
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirNetSearchADAPDirectoriesParse
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirRemoveDirectory
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetOCESetupRefNum
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
```

Creating, Opening, and Closing Personal Catalogs

```
FUNCTION DirCreatePersonalDirectory
                            (paramBlock: DirParamBlockPtr): OSErr;
FUNCTION DirOpenPersonalDirectory
                            (paramBlock: DirParamBlockPtr): OSErr;
```

```
FUNCTION DirClosePersonalDirectory
                            (paramBlock: DirParamBlockPtr): OSErr;
FUNCTION DirMakePersonalDirectoryRLI
                            (paramBlock: DirParamBlockPtr): OSErr;
```

Managing Records

```
FUNCTION DirAddRecord       (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirDeleteRecord    (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetRecordMetaInfo
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetNameAndType  (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirSetNameAndType  (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirAddPseudonym    (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirDeletePseudonym
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirEnumeratePseudonymGet
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirEnumeratePseudonymParse
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirAddAlias        (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
```

Managing Attribute Types and Values

```
FUNCTION DirAddAttributeValue
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirDeleteAttributeValue
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirChangeAttributeValue
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
```

```
FUNCTION DirVerifyAttributeValue
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirFindValue       (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirLookupGet       (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirLookupParse     (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirDeleteAttributeType
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirEnumerateAttributeTypesGet
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirEnumerateAttributeTypesParse
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
```

Reading Access Controls for dNodes, Records, and Attribute Types

```
FUNCTION OCEGetAccessControlDSSpec
                            (categoryBitMask: CategoryMask): DSSpecPtr;
FUNCTION DirGetDNodeAccessControlGet
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetDNodeAccessControlParse
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetRecordAccessControlGet
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetRecordAccessControlParse
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetAttributeAccessControlGet
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetAttributeAccessControlParse
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
```

Canceling a Catalog Manager Function

```
FUNCTION DirAbort           (paramBlock: DirParamBlockPtr): OSErr;
```

Application-Defined Functions

```
FUNCTION MyCompletionRoutine
                            (paramBlk: DirParamBlockPtr);
FUNCTION MyForEachRecordID  (clientData: long;
                             recordID: RecordID): BOOLEAN;
FUNCTION MyForEachAttrType  (clientData: long;
                             attrType: AttributeType): BOOLEAN;
FUNCTION MyForEachDirectory
                            (clientData: long; dirName: DirectoryNamePtr;
                             discriminator: DirDiscriminator;
                             features: DirGestalt): BOOLEAN;
FUNCTION MyForEachLookupRecordID
                            (clientData: long;
                             recordID: RecordID): BOOLEAN;
FUNCTION MyForEachAttrTypeLookup
                            (clientData: long; attrType: AttributeTypePtr;
                             myAttrAccMask: AccessMask): BOOLEAN;
FUNCTION MyForEachAttrValue (clientData: long;
                             attribute: Attribute): BOOLEAN;
FUNCTION MyForEachDirEnumSpec
                            (clientData: LONGINT;
                             enumSpec: DirEnumSpec): BOOLEAN;
FUNCTION MyForEachRecord    ((clientData: long;
                             dsObj: DSSpec; activeDnodeAccMask: AccessMask;
                             activeRecordAccMask: AccessMask;
                             defaultAttributeAccMask: AccessMask): BOOLEAN;}
FUNCTION MyForEachADAPDirectory
                            (clientData: long; dirName: DirectoryNamePtr;
                             discriminator: DirDiscriminator;
                             features: DirGestalt; serverHint: AddrBlock):
                             BOOLEAN;
FUNCTION MyForEachDNodeAccessControl
                            (clientData: long; dsObj: DSSpec;
                             activeDnodeAccMask: AccessMask;
                             defaultRecordAccMask: AccessMask;
                             defaultAttributeAccMask: AccessMask):BOOLEAN;
```

```
FUNCTION MyForEachRecordAccessControl
                              (clientData: long; dsObj: DSSpec;
                               activeDnodeAccMask: AccessMask;
                               activeRecordAccMask: AccessMask;
                               defaultAttributeAccMask: AccessMask):BOOLEAN;
FUNCTION MyForEachAttributeAccessControl
                              (clientData: long; dsObj: DSSpec;
                               activeDnodeAccMask: AccessMask;
                               activeRecordAccMask: AccessMask;
                               activeAttributeAccMask: AccessMask): BOOLEAN;
```

Assembly-Language Summary

Trap Macros Requiring Routine Selectors

`_oceTBDispatch`

Selector	Routine
0x101	DirEnumerateParse
0x102	DirLookupParse
0x103	DirEnumerateAttributeTypesParse
0x104	DirEnumeratePseudonymParse
0x105	DirNetSearchADAPDirectoriesParse
0x106	DirEnumerateDirectoriesParse
0x107	DirFindADAPDirectoryByNetSearch
$0108	`DirNetSearchADAPDirectoriesGet`
$0109	`DirAddRecord`
$010A	`DirDeleteRecord`
$010B	`DirAddAttributeValue`
$010C	`DirDeleteAttributeValue`
$010D	`DirChangeAttributeValue`
$010E	`DirVerifyAttributeValue`
$010F	`DirAddPseudonym`
$0110	`DirDeletePseudonym`
$0111	`DirEnumerateGet`
$0112	`DirEnumerateAttributeTypesGet`
$0113	`DirEnumeratePseudonymGet`
$0114	`DirGetNameAndType`
$0115	`DirSetNameAndType`
$0116	`DirGetRecordMetaInfo`

Selector	Routine
$0117	DirLookupGet
$0118	DirGetDNodeMetaInfo
$0119	DirGetDirectoryInfo
$011A	DirEnumerateDirectoriesGet
$011B	DirAbort
$011C	DirAddAlias
$011D	DirAddDSAM
$011E	DirOpenPersonalDirectory
$011F	DirCreatePersonalDirectory
$0121	DirGetDirectoryIcon
$0122	DirMapPathNameToDNodeNumber
$0123	DirMapDNodeNumberToPathName
$0124	DirGetLocalNetworkSpec
$0125	DirGetDNodeInfo
$0126	DirFindValue
$0128	DirGetOCESetupRefNum
$012A	DirGetDNodeAccessControlGet
$012C	DirGetRecordAccessControlGet
$012E	DirGetAttributeAccessControlGet
$012F	DirGetDNodeAccessControlParse
$0130	DirDeleteAttributeType
$0131	DirClosePersonalDirectory
$0132	DirMakePersonalDirectoryRLI
$0134	DirGetRecordAccessControlParse
$0135	DirRemoveDirectory
$0136	DirGetExtendedDirectoriesInfo
$0137	DirAddADAPDirectory
$0138	DirGetAttributeAccessControlParse
$0140	DirFindRecordGet
$0141	DirFindRecordParse

Result Codes

The allocated range of result codes for the Catalog Manager is –1610 through –1646 and there are some result codes in the range –1503 through –1567. Functions may also return result codes from other AOCE managers and standard Macintosh result codes such as `noErr` 0 (No error) and `fnfErr` –43 (File not found).

`kOCEBufferTooSmall`	–1503	Buffer too small for data requested
`kOCEVersionErr`	–1504	Need to sort personal catalog
`kOCEAlreadyExists`	–1510	The catalog being added already exists
`kOCEReadAccessDenied`	–1540	Identity lacks read access privileges
`kOCEWriteAccessDenied`	–1541	Identity lacks write access privileges
`kOCEUnknownID`	–1567	Authentication identity is not valid
`kOCENotLocal`	-1610	The server does not serve the requested dNode
`kOCETooBusy`	-1611	Server cannot complete call at this time
`kOCEDatabaseFull`	-1612	The disk is full
`kOCETargetDirectoryInaccessible`	-1613	Target catalog is not currently available
`kOCEBogusArgs`	-1614	Args not formatted correctly on the wire
`kOCENoSuchDNode`	-1615	Can’t find specified dNode
`kOCEDNodeUnavailable`	-1616	Could not find any servers that serve the requested dNode
`kOCEBadRecordID`	-1617	Record name or record type doesn’t match creation ID
`kOCENoSuchRecord`	-1618	Can't find specified record
`kOCENoSuchAttributeValue`	-1619	Can’t find specified attribute value
`kOCENoSuchPseudonym`	-1620	The specified pseudonym does not exist
`kOCEAttributeValueTooBig`	-1621	Attribute value is larger than `kAttrValueMaxBytes` bytes
`kOCETypeExists`	-1622	The type already exists in the record
`kOCEMoreData`	-1623	More data available
`kOCERefNumBad`	-1624	RefNum is not valid
`kOCEStreamCreationErr`	-1625	Error in creating connection to server
`kOCEOperationNotSupported`	-1626	The specified catalog does not support this operation
`kOCEPABNotOpen`	-1627	The specified personal catalog is not open to make the operation
`kOCEDSAMInstallErr`	-1628	The specified CSAM could not be installed
`kOCEDirListFullErr`	-1629	The catalog list is full; try removing an entry
`kOCEDirectoryNotFoundErr`	-1630	Can’t find catalog
`kOCEAbortNotSupportedForThisCall`	-1631	Abort not supported

kOCEAborted	-1632	The call was aborted
kOCEOCESetupRequired	-1633	LocalIdentity Setup is required
kOCEDSAMRecordNotFound	-1634	CSAM Record not found
kOCEDSAMNotInstantiated	-1635	CSAM is not instantiated
kOCEDSAMRecordExists	-1636	CSAM record already exists
kOCELengthError	-1637	The buffer supplied was too small
kOCEBadStartingRecord	-1638	Starting record index out of range
kOCEBadStartingAttribute	-1639	Starting attribute index is not within range
kOCEMoreAttrValue	-1640	Buffer too small for a single attribute value
kOCENoDupAllowed	-1641	Duplicate name and type
kOCENoSuchAttributeType	-1642	Can't find specified attribute type
kOCEMiscError	-1643	Miscellaneous error
kOCENoSuchIcon	-1644	There is no matching icon from OCEGetDirectoryIcon
kOCERLIsDontMatch	-1645	RLIs of different records in the record list are not the same
kOCEDirectoryCorrupt	-1646	Serious disk fill corruption problem